This Page Blank (uspto)

ℷ.९ɾ

ELSEVIER

Speech Communication 16 (1995) 89–118

SPEECH COMMUNICATION

Bayesian Belief Networks as a tool for stochastic parsing [*]

Helmut Lucke

ATR Interpreting Telecommunications Research Laboratories, 2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto, Japan

Received 25 February 1994; revised 18 August 1994

Abstract

Bayesian Belief Networks are a powerful tool for combining different knowledge sources with various degrees of uncertainty in a mathematical sound and computationally efficient way. Surprisingly they have not yet found their way into the speech processing field, despite the fact that in this science multiple unreliable information sources exist. The present paper shows how the theory can be utilized in for language modeling. After providing an introduction to the theory of Bayesian Networks, we develop several extensions to the classic theory by describing mechanisms for dealing with statistical dependence among daughter nodes (usually assumed to be conditionally independent) and by providing a learning algorithm based on the EM-algorithm with which the probabilities of link matrices can be learned from example data. Using these extensions a language model for speech recognition based on a context-free framework is constructed. In this model, sentences are not parsed in their entirety, as is usual with grammatical description, but only "locally" on suitably located segments. The model was evaluated over a text data base. In terms of test set entropy the model performed at least as good as the bi/tri-gram models, while showing a good ability to generalize from training to test data.

Zusammenfassung

Bayessche Belief Netzwerke sind nützliche Werkzeuge um auf mathematisch rigorose rechnerisch nicht zu aufwendige Weise verschiedene Wissensquellen mit unterschiedlichem Grad an Unsicherheit miteinander zu verbinden. Erstaunlicherweise werden sie bis heute kaum in der Spracherkennung angewendet, obwohl gerade in diesem Gebiet verschieden unsichere Wissensquellen betrachtet und ausgenutzt werden müssen um brauchbare Erkennungsraten zu erzielen. Nach einer Einführung in die Theorie der Bayesschen Netze werden einige für die Sprachmodellierung notwendige Erweiterungen beschrieben. Insbesondere wird auf die statistische Unabhängigkeit der direkten Folgen einer Ursache verzichtet und ein Lernverfahren basierend auf dem EM-Algorithmus beschrieben. Hiermit können die Modellparameter auch bei unvollständigen Trainingsmaterial gelernt werden. Mit Hilfe dieser Erweiterungen ist es möglich ein Sprachmodell basierend auf stochastischen Kontext-freien Grammatikregeln zu realisieren. Ein solches Model wird beschrieben und experimentell auf einer Textdatenbank evaluiert. Hierbei zeigte das Model einen mindestens so guten Sprach-Entropie wie die n-gram Modelle und bewies eine gute Fähigkeit vom Trainingsmaterial auf das Testmaterial zu verallgemeinern.

[*] This paper is based on a communication presented at the ESCA Conference EUROSPEECH-93 and has been recommnended by the EUROSPEECH-93 scientific program committee.

Résumé

Les réseaux bayésiens de modélisation de la croyance constituent un outil efficace pour combiner différentes sources d'information ayant différents degrés d'incertitude. Ils sont rigoureux mathématiquement et peuvent être implémentés de façon efficace. Ils sont malheureusement peu utilisés en traitement de la parole. Cet article montre comment ils peuvent être appliqués à ce domaine. Après une introduction aux réseaux bayésiens, nous proposons certaines extensions de la théorie classique. Nous décrivons comment rendre compte de la dépendance statistique entre les noeuds-fils, d'habitude considérés comme indépendants. Nous donnons aussi un algorithme, inspiré de l'algorithme EM, qui permet l'apprentissage de matrices de liens à partir d'exemples. A l'aide de ces extensions, nous construisons un modèle de langage à base "context-free" pour la reconnaissance de la parole, dans lequel les phrases sont analysées localement, en segments judicieusement choisis, au lieu d'être analysées dans leur totalité. Ce modèle a été évalué sur une base de données textuelles. Sa performance, mesurée par l'entropie de l'ensemble de test, est au minimum égale à celle des modèles bi- ou tri-grammes; nous avons aussi constaté une bonne capacité de généralisation des données d'apprentissage vers les données de test.

Keywords: Bayesian Networks; Grammar inference; Stochastic parsing

1. Introduction

Robust automatic speech recognition requires the use, and hence combination, of several uncertain information sources. Firstly the acoustic signal of an utterance, being constrained by the physical movements of the articulators and contaminated with noise, contains a high degree of uncertainty. Even after processing by acoustic models such as hidden Markov models the degree of uncertainty in lists of candidate words or word lattices remains high. Language models are used to further improve the selection of the word strings. These models are usually themselves expressed probabilistically. Using semantic and pragmatic information to further assist the speech recognition process has also been proposed and again such a scheme would rely on the combination of various uncertain information sources.

As for language models, the n-gram model has been remarkably successful (Jelinek, 1991). Most popular are the so-called bigram and trigram models, although through a tree-based clustering method it has been possible to extend these to 21-grams (Bahl et al., 1989). It is somehow surprising that without performing any structural analysis, as advocated by traditional linguistics, such models can perform so well. Their remarkable success can only be explained by their solid (if simple) mathematical basis.

In this paper we propose a language model which is based on structural analysis. This can become useful when structural analysis is required to extract meaning or to interact with semantic or even pragmatic knowledge sources. The basic formalism adopted is based on Bayesian belief networks, a stochastic inference mechanism mainly developed for probabilistic expert systems. It is anticipated that this formalism will make it easier to incorporate further knowledge sources, although this is not attempted in this paper. Here we concentrate mainly on the theoretical background necessary in adopting Bayesian belief networks for linguistic analysis. We will describe an iterative training method that learns both structure and probabilities in an unsupervised fashion. Convergence of the algorithm will be shown.

From a linguistic view point, the algorithm assumes context-freeness of the language and learns probabilistic production rules that are best supported by the training data. As such, the algorithm falls in the class of grammatical inference algorithms. Prior work in this area includes many symbolic (e.g. (Anderson, 1981; Berwick, 1980; Wolff, 1980, 1982)), connectionist (McClelland and Rumelhart, 1987) and probabilistic (Baker, 1979) approaches. A good survey of earlier work was written by Fu and Booth

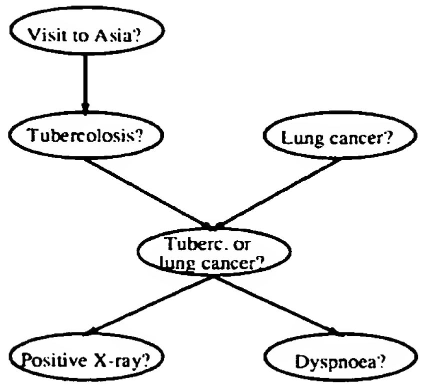


Fig. 1. A typical Bayesian network for use in a medical expert system (after (Cowel et al., 1993)).

(1986). Some limits on the learnability of unconstrained context-free languages in the non-probabilistic case are also known (Gold, 1967).

2. Bayesian Belief Networks

Bayesian Networks (influence diagrams) are a tool for calculating posterior probability distributions over sets of random variables. For a good treatment see (Pearl, 1987). They have been studied in the artificial intelligence literature with the aim of producing expert systems that are capable of dealing with uncertain information. For example in a medical application a Bayesian Network may relate observable symptoms to unobservable causes (i.e. physiological conditions). A typical such network is shown in Fig. 1.

In such a network each node represents a random variable. Pearl notes that if the underlying graph is a tree then there exists a simple algorithm for calculating the posterior probability distribution of each variable in the network. The aim of this paper is to show that this theory can easily be applied to statistical modeling in speech where we similarly have observable "symptoms" such as the waveform or surface word string and unobservable (or hidden) variables such as HMM states or the linguistic units (like "noun phrase", etc).

A typical Bayesian network in the form of a tree is shown in Fig. 2. Each node corresponds to a random variable. For simplicity we will assume that they are all discrete. Some of the variables may be observable, others may not be. For a given observation, the states of the observable nodes are called the evidence and are denoted e. The arrows in the graph indicate the assumed causal influences. Thus an arrow pointing from node A to node B means that A has a direct causal influence on B. This relationship is quantified in a matrix $\Pr(B \mid A)$ which gives for each possible value of A the probability distribution of B. Our knowledge base then consists of the directed graph, the link matrices for each link and the prior probability distribution $\Pr(R)$ of the root node. Given this knowledge base and an observation (instantiation of the observable nodes) we would then like to calculate the posterior probability distribution of all the unobservable nodes A, i.e. find the distributions $\Pr(A \mid e)$.

Pearl describes a simple and efficient algorithm for calculating these probability distributions. It is based on the propagation of two vectors known as the *diagnostic* and *causal* support vectors through the network.

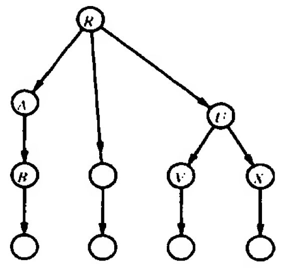


Fig. 2. A typical causal tree diagram.

In the remainder of this introduction we will define the notation and quote the propagation equations. In Section 4 we will generalize them in a form suitable for the experimental work and will supply the necessary derivations. We will also provide a training method for learning the link matrices automatically even between unobserved nodes. In Section 5 the convergence of the training method will be proved. We will then describe how the presented ideas could be used for learning a language model based on a context-free paradigm (Section 6) and report experimental results (Section 7).

Conditional probabilities: The derivations in this paper make use of conditional probabilities of the form $\Pr(A|B)$. We will use capital letters A, B, C to indicate nodes in the networks. A node A can be thought of as a discrete random variable taking values $a_1, a_2, \ldots$. If one or more of these node variables enters the argument of the probability function P the resulting expression is to be interpreted as a tensor. For example the expression $\Pr(e_U^- \mid U)$ is a vector. It has as many components as U can take values. The i'th component is $\Pr(e_U^- \mid U = u_i)$. Similarly $\Pr(e_U^-, U \mid V, X, e_X^+)$ is a tensor of rank 3 whose ijk'th component is $\Pr(e_U^-, U = u_i \mid V = v_j, X = x_k, e_X^+)$.

Important vectors and their definitions:

$\lambda(U)$	$= \Pr(e_U^- \mid U)$	(diagnostic support vector)
$\pi(U)$	$= \Pr(U \mid e_U^+)$	(causal support vector)
$\mathrm{BEL}(U)$	$= \Pr(U \mid e)$	(Belief vector)

Vector products:

$\boldsymbol{ab}$	componentwise vector product: $(\boldsymbol{ab})_i = a_i b_i$
$\boldsymbol{a} \cdot \boldsymbol{b}$	familiar dot product: $\boldsymbol{a} \cdot \boldsymbol{b} = \sum_i a_i b_i$
$\lambda(U)\Pr(U \mid V)$	vector-matrix product. Vector matrix product of this form are performed in the only sensible way, i.e. identifying states of the same variable. So if i indices states of the variable U and j indices states of the variable V then $\lambda(U)\Pr(U\|V)_j = (\sum_i \lambda(U)_i \Pr(U{=}i\|V{=}j))$. Thus in general we will not make transpositions of vectors or matrices explicit.

Normalizing constants: When the letter α appears in front of a vector it represents a real number normalizing that vector. Occasionally α occurs more than once in an equation. In this case they usually represent different normalization factors.

Fig. 3. Notation used in this paper.

2.1. Belief calculation by graph propagation equations

We will begin by defining some notation. Random variables will be written by upper case letters. These usually correspond to the nodes in the graph and will be identified with these. The evidence e stands for the total observed information (values of the observed random variables). Following Pearl, for a node X we write e_X^- to indicate the part of the evidence that is connected to one of the descendants of X (i.e. that can be reached from X by walking only in the direction of the arrows in the graph) and e_X^+ the remaining evidence $e \backslash e_X^-$. Some additional notation is summarized in Fig. 3.

Furthermore we define for each node X of the diagram two vectors: The diagnostic support vector,

$$\lambda(X) = \Pr(e_X^- \mid X) = (\Pr(e_X^- \mid X = x_i)), \tag{1}$$

and the causal support vector,

$$\pi(X) = \Pr(X \mid e_X^+) = (\Pr(X = x_i \mid e_X^+)). \tag{2}$$

The essence of the propagation theory lies in the fact that these quantities can be "propagated" through the underlying graph and thereby calculated recursively. In order to state these recursions we define two auxiliary vectors. If V is a parent of nodes $U_1, \ldots, U_k$ (see Fig. 4), we define

$$\lambda_{U_r}(V) = \Pr(e_{U_r}^- \mid V), \tag{3}$$

$$\pi_{U_r}(V) = \Pr(V \mid e_{U_r}^+). \tag{4}$$

One can now show (for a derivation see (Pearl, 1987) or Section 4.1 where we derive more general versions of these equations):

$$\lambda_{U_r}(V) = \lambda(U_r)\Pr(U_r \mid V), \tag{5}$$

$$\lambda(V) = \prod_{r=1}^{k} \lambda_{U_r}(V), \tag{6}$$

$$\pi_{U_r}(V) = \alpha\pi(V)\prod_{l \neq r} \lambda_{U_l}(V), \tag{7}$$

$$\pi(U_r) = \Pr(U_r \mid V)\pi_{U_r}(V). \tag{8}$$

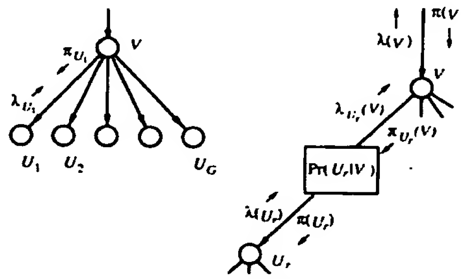


Fig. 4. Illustration of the propagation mechanism.

Here the products on the right-hand-side of Eqs. (5) and (8) are familiar vector–matrix products. The vector product involved in Eqs. (6) and (7) is the component-wise vector product (i.e. vector × vector = vector). The coefficient α in Eq. (7) is a scalar chosen such that the vector $\pi_{U_r}(V)$ on the left-hand-side is normalized.

Using Eqs. (6) and (8) it is thus possible to calculate the λ and π vectors of all nodes in the network from the π vector of the root node (node H in our example) and the λ vectors of the other leaf nodes. The posterior probability distribution $\Pr(U \mid e)$ of a node U is denoted BEL(U) and can be calculated as

$$\mathrm{BEL}(U) = \Pr(U \mid e) = \alpha\lambda(U)\pi(U) = \frac{\lambda(U)\pi(U)}{\lambda(U)\cdot\pi(U)}, \tag{9}$$

where again the scalar α is chosen such as to normalize the resulting vector.

3. Analysis of Bayesian belief propagation

In the previous section we stated the propagation equations for Bayesian networks. Since the calculations are local at each node, it is clear that even complex dependencies can be calculated easily and accurately by following the structure of the tree. The theory as described above is only applicable to simple trees, i.e. causal structures in which each event can have at most one cause, however a generalization to poly-trees, where multiple causes exist for an event, is also available, but not discussed in this paper.

How could an expert system make use of this approach for automated reasoning? According to the ideas put forward in the previous section, the knowledge base of such a system would consist of probability matrices of the form $\Pr(Y \mid X)$ for events X and Y known or observed to be in a causal relationship and prior distributions $\Pr(X)$. For a given problem it would then be the task of the system to propose a plausible network structure connecting the observed events with other unobserved but relevant events. After a network is constructed, the belief propagation equations can be used to calculate the posterior probability distribution of the unobserved nodes of interest.

Thus the expert system has to cope with two different problems: a qualitative one, consisting of the construction of the graphical structure and a quantitative one propagating the λ and π vectors through this network.

Expert systems implementing the quantitative part have already been proposed by Lauritzen and Spiegelhalter (1988) and others, however these systems are not able to solve the first problem: the assignment of the graphical structure. Instead this structure is given in advance and hence forms part of the knowledge base.

While the studies on such expert systems are interesting and help to demonstrate the propagation mechanism, they are of limited use in practice. It is unfeasible for an expert system to maintain a very large tree connecting all possible (observed and unobserved) events. Such a network would connect seemingly unrelated events. Moreover, it would require the propagation of the λ and π vectors from parts of the network that are only marginally related to the events of interest. It would also be extremely difficult to establish a large network connecting all nodes of interest and yet have it loop-free as required by the theory.

Instead one should look for an approach that constructs a network "on the fly" connecting nodes when they become relevant in the light of observed events. This would entail establishing the relevance of various observed and unobserved events on the basis of the known $\Pr(Y \mid X)$ matrices and constructing a network which connects the unobserved events of interest to the relevant observed events, possibly creating new (previously unknown) unobserved events in the process. We are not aware of any such algorithm reported in the literature.

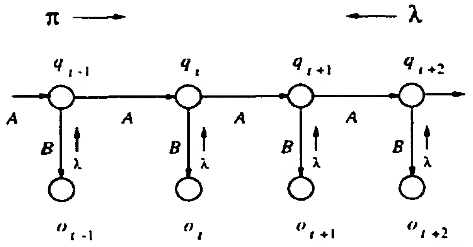


Fig. 5. Bayesian tree interpretation of an HMM.

In Section 6 we use the theory of Belief trees to stochastically parse a sentence. In this paradigm the words of the sentence form the observed events. Unobserved events are grammatical markers such as "noun phrase", "prepositional phrase", etc. It is the object of the parser to connect the possible unobserved events to the observed events in the "best possible way". This will create a tree structure (the parse tree). The theory of Bayesian networks will give us the quantitative tool for calculating the beliefs of all unobserved events and also the likelihood of the chosen tree structure. Thus the problem of finding the best parse tree will be the same as finding the best causal explanation of the observed evidence, i.e. it directly corresponds to the problem described in the previous paragraph for expert systems.

3.1. Relationship to hidden Markov Models

Hidden Markov Models can be viewed as Bayesian trees of a certain fixed topology. Fig. 5 shows such an interpretation.

A number of events (the observation sequence) $o_1, \ldots, o_t, \ldots$ is observed. In addition, unobservable variables $q_1, \ldots, q_t, \ldots$ (describing the state occupancy at time t) are assumed to exist. The probabilistic dependencies between the variables are shown by the arrows and quantified by the two matrices A and B (assuming a discrete HMM). The λ and π vectors take the roles of the familiar backward and forward probabilities, respectively. The theory of Bayesian Networks is directly applicable because the structure of the network is fixed as shown.

In contrast, the Inside–Outside algorithm, a grammatical inference algorithm proposed by Baker (1979), does not operate on Bayesian networks. Because of its similarity to the grammar inference algorithm described in this paper we will discuss this relationship in more detail in Section 6.11.

4. Extensions to the basic belief propagation equations

With regard to Section 6.2, we will now present a few extensions to the belief propagation equations. We will also provide the necessary derivations of all results. Since the equations quoted in the previous section are special cases of the results developed here, the proofs apply to the previous section as well.

4.1. Dependence of daughter nodes dependencies

In the diagram we discussed in the introduction, the daughters of a given parent node were assumed to be conditionally independent (meaning independent once the value of the parent is known). Mathematically this is expressed for parent U with daughters V and X as

$$\Pr(V,X \mid U=u) = \Pr(V \mid U=u)\Pr(X \mid U=u), \tag{10}$$

for each possible u.

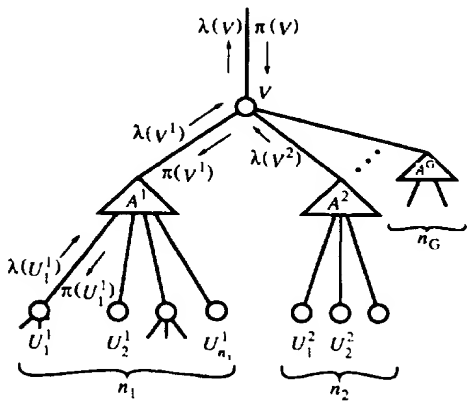


Fig. 6. This figure shows a node V with daughters $U_1^1, U_2^1, \ldots, U_{n_1}^1, U_1^2, \ldots, U_{n_2}^2, \ldots, U_{n_G}^G$). The daughter nodes are divided into G groups as shown. The groups are regarded as statistically independent, but the nodes within each group are dependent. Hence the conditional probability distribution of the daughters given the parent has the product form shown in Eq. (11).

The belief propagation equations can be modified to apply in situations when Eq. (10) does not hold. If there are dependency relations among the daughter nodes, it is not sufficient to specify each parent–daughter relation separately, but rather jointly using a tensor of rank 3 or higher. This tensor expresses the joint conditional probabilities between daughter nodes $U_1, \ldots, U_n$ and parent V, viz. $\Pr(U_1, \ldots, U_n \mid V)$ instead of separate link matrices $\Pr(U_1 \mid V), \ldots, \Pr(U_n \mid V)$. In the most general case, the daughters of V can be divided into, say, G groups. Nodes within one group are regarded as statistically independent from the nodes of any other group, so there is no need to specify the parent–daughter relation by one large tensor. However, the nodes within each group are considered to be statistically dependent and their relation to the parent is expressed by a tensor for this group. Thus if the nodes in group g are labeled $U_1^g, \ldots, U_{n_g}^g$ the overall conditional probability tensor has the form

$$\Pr\left(U_1^1, \ldots, U_{n_1}^1, U_1^2, \ldots, U_{n_2}^2, \ldots, U_1^G, \ldots, U_{n_G}^G \mid V\right) = \prod_{g=1}^{G} \Pr\left(U_1^g, U_2^g, \ldots, U_{n_g}^g \mid V\right). \tag{11}$$

Fig. 6 displays such a scenario graphically.

The propagation equations can be adopted in a straightforward way to the new situation. For convenience we define some additional notation. We denote the group consisting of nodes $U_1^g, \ldots, U_{n_g}^g$ by V^g. Further we define the evidence $e_{V^g}^-$ as the combined evidence $e_{U_1^g}^-, \ldots, e_{U_{n_g}^g}^-$ and likewise $e_{V^g}^+$ as the remaining evidence $e \setminus e_{V^g}^-$, i.e. the entire evidence e less $e_{V^g}^-$. Furthermore we define

$$\lambda(V^g) = \Pr(e_{V^g}^- \mid V), \tag{12}$$

$$\pi(V^g) = \Pr(V \mid e_{V^g}^+), \tag{13}$$

and we write A^g to indicate the tensor for group g, i.e.

$$A_{i, j_1, \ldots, j_{n_g}}^g = \Pr\left(U_1^g = j_1, \ldots, U_{n_g}^g = j_{n_g} \mid V = i\right). \tag{14}$$

4.2. Applicable laws of probability

In order to derive Eqs. (5) to (8) and their generalizations we require three laws of probability theory. These are described below.

4.2.1. Bayes' law

Bayes' law states that for events a, b, c we have

$$\Pr(a \mid b,c) = \frac{\Pr(b \mid a,c)\Pr(a \mid c)}{\Pr(b \mid c)}. \tag{15}$$

Now, let U be a random variable corresponding to a node of the network, so that in our notation $\Pr(U \mid b, c)$ is a vector. This vector must be normalized. We can then write Bayes' law in the following form:

$$\Pr(U \mid b,c) = \alpha \Pr(b \mid U,c)\Pr(U \mid c), \tag{16}$$

where $\alpha = 1/\Pr(b \mid c)$ can be determined simply as a normalizing constant.

4.2.2. Conditioning

Let a and b be events and U a random variable that can take a finite number of values $1, \dots, n$. The following equation holds:

$$\Pr(a \mid b) = \sum_i \Pr(a \mid U,b)_i \Pr(U \mid b)_i. \tag{17}$$

4.2.3. Separation

To illustrate the law of separation consider the probability vector $\Pr(e_U^- \mid e_U^+, U)$. Since the graphical structure is assumed to be a tree, the value of e_U^+ can effect the value of e_U^- only via the value of U. We say e_U^+ and e_U^- are marginally independent, i.e independent when conditioned on the value of U. Hence

$$\Pr(e_U^- \mid e_U^+, U) = \Pr(e_U^- \mid U). \tag{18}$$

We also say "U separates e_U^+ from e_U^-".

4.3. Derivation of the propagation equations

We will begin by proving a generalization of Eq. (5). The capital letters B, C, S and D on top of the equal sign (=) indicate applications of Bayes' Law, the laws of Conditioning and Separation and a definition, respectively. A bracketed numeral indicates an application of an earlier equation. Furthermore, for readability, we will identify a value x_i of a random variable X with the index i itself, writing $\Pr(X = i)$ instead of $\Pr(X = x_i)$. In fact, in Eq. (14), we already used this notation.

Now, as for the λs we have

$$\lambda(V^g)_i \overset{\text{D}}{=} \Pr(e_{V^g}^- \mid V = i) \tag{19}$$

$$\overset{\text{D}}{=} \Pr\left(e_{U_1^g}^-, \dots, e_{U_{n_g}^g}^- \mid V = i\right) \tag{20}$$

$$\overset{\text{C}}{=} \sum_{j_1, \dots, j_{n_g}} \Pr\left(e_{U_1^g}^-, \dots, e_{U_{n_g}^g}^- \mid V = i,\, U_1^g = j_1, \dots U_{n_g}^g = j_{n_g}\right) A_{i,j_1,\dots,j_{n_g}}^g \tag{21}$$

$$\overset{\text{S}}{=} \sum_{j_1, \dots, j_{n_g}} \prod_{k=1}^{n_g} \Pr\left(e_{U_k^g}^- \mid U_k^g = j_k\right) A_{i,j_1,\dots,j_{n_g}}^g \tag{22}$$

$$\overset{\text{D}}{=} \sum_{j_1, \dots, j_{n_g}} \prod_{k=1}^{n_g} \lambda(U_k^g)_{j_k} A_{i,j_1,\dots,j_{n_g}}^g, \tag{23}$$

$$\lambda(V)_i \overset{\text{D}}{=} \Pr(e^-_V \mid V=i) \tag{24}$$

$$\overset{\text{D}}{=} \Pr(e^-_{V^1},\ldots,e^-_{V^G} \mid V=i) \tag{25}$$

$$= \prod_{g=1}^{G} \Pr(e^-_{V^g} \mid V=i) \tag{26}$$

$$\overset{\text{D}}{=} \prod_{g=1}^{G} \lambda(V^g)_i. \tag{27}$$

Eq. (23) expresses the vector $\lambda(V^g)$ as a simple tensor–vector product between the tensor A^g and the λ-vectors $\lambda(U^g_k)$, $k=1,\ldots,n_g$. Regarding Fig. 6 one could say that the vectors $\lambda(U^g_k)$ enter the triangle (tensor) A^g from the bottom. Here the tensor–vector product is formed and the λ vector representing the group $\lambda(V^g)$ is emitted at the top. Eq. (27) shows how the λ vectors from the various groups need to be combined to give the overall λ vector: by component-wise vector product. One could write these two results in the vector form:

$$\lambda(V^g) = A^g_{i,j_1,\ldots,j_{n_g}} \lambda(V^g_1) \cdots \lambda(V^g_{n_g}), \tag{28}$$

$$\lambda(V) = \prod_{g=1}^{G} \lambda(V^g), \tag{29}$$

where the multiplication in Eq. (28) is a tensor–vector product (like a generalized matrix–vector product) and the multiplication in Eq. (29) is a componentwise vector product. However, since it is difficult to distinguish between the two forms of multiplication in this notation and it is also not easy to see which indices of the tensor A^g identify with which λ vector in Eq. (28), we will always use the more explicit component-wise description shown in Eqs. (23) and (27).

For the πs we derive

$$\pi(V^g)_i \overset{\text{D}}{=} \Pr(V=i \mid e^+_{V^g}) \tag{30}$$

$$\overset{\text{D}}{=} \Pr\left(V=i \mid e^+_V,\, e^-_{V^1},\ldots,\cancel{e^-_{V_g}},\ldots,e^-_{V^g}\right) \tag{31}$$

$$\overset{\text{B}}{=} \alpha\ \Pr\left(V=i,\, e^-_{V^1},\ldots,\cancel{e^-_{V_g}},\ldots,e^-_{V^G} \mid e^+_V\right) \tag{32}$$

$$\overset{\text{C}}{=} \alpha\ \Pr(v=i \mid e^+_V)\ \Pr\left(V=i,\, e^-_{V^1},\ldots,\cancel{e^-_{V_g}},\ldots,e^-_{V^G} \mid e^+_V,\, V=i\right) \tag{33}$$

$$\overset{\text{S}}{=} \alpha\ \Pr(V=i \mid e^+_V) \prod_{g'=1,\ldots,\cancel{g},\ldots,G} \Pr(e^-_{V^{g'}} \mid V=i) \tag{34}$$

$$\overset{\text{D}}{=} \alpha\pi(V)_i \prod_{g'=1,\ldots,\cancel{g},\ldots,G} \lambda(V^{g'})_i, \tag{35}$$

$$\pi(U^g_k)_{j_k} \overset{\text{D}}{=} \Pr(U^g_k = j_k \mid e^+_{U^g_k}) \tag{36}$$

$$\overset{\text{D}}{=} \Pr\left(U^g_k = j_k \mid e^+_{V^g},\ \bigwedge_{k'\neq k} e^-_{U^g_{k'}}\right) \tag{37}$$

$$\overset{\text{C}}{=} \sum_{i,j_1,\ldots,\not{j}_k,\ldots,j_{n_g}} \left\{ \Pr\left(U_k^g = j_k \mid V=i, \left(\bigwedge_{k'\neq k} U_{k'}^g = j_{k'}\right), e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g}\right) \times \Pr\left(V=i, \bigwedge_{k'\neq k} U_{k'}^g = j_{k'} \,\middle|\, e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g}\right)\right\} \tag{38}$$

$$\overset{\text{S,B}}{=} \alpha_1 \sum_{i,j_1,\ldots,\not{j}_k,\ldots,j_{n_g}} \left\{ \Pr\left(U_k^g = j_k \mid V=i, \bigwedge_{k'\neq k} U_{k'}^g = j_{k'}\right) \Pr(V=i) \times \Pr\left(\bigwedge_{k'\neq k} U_{k'}^g = j_{k'}, e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g} \,\middle|\, V=i\right)\right\} \tag{39}$$

$$\overset{\text{B}}{=} \alpha_1 \sum_{i,j_1,\ldots,\not{j}_k,\ldots,j_{n_g}} \left\{ \frac{\Pr\left(\bigwedge_{k'=1,\ldots,n_g} U_{k'}^g = j_{k'} \mid V=i\right)}{\Pr\left(\bigwedge_{k'\neq k} U_{k'}^g = j_{k'} \mid V=i\right)} \Pr(V=i) \times \Pr\left(\bigwedge_{k'\neq k} U_{k'}^g = j_{k'}, e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g} \,\middle|\, V=i\right)\right\} \tag{40}$$

$$\overset{\text{D,B}}{=} \alpha_1 \sum_{i,j_1,\ldots,\not{j}_k,\ldots,j_{n_g}} A^g_{i,j_1,\ldots,j_{n_g}} \Pr(V=i) \Pr\left(e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g} \,\middle|\, \bigwedge_{k'\neq k} U_{k'}^g = j_{k'}, V=i\right) \tag{41}$$

$$\overset{\text{S}}{=} \alpha_1 \sum_{i,j_1,\ldots,\not{j}_k,\ldots,j_{n_g}} A^g_{i,j_1,\ldots,j_{n_g}} \Pr(V=i) \Pr(e^+_{V^g} \mid V=1) \prod_{k'=1,\ldots,\not{k},\ldots,n_g} \Pr\left(e^-_{U_{k'}^g} \mid U_{k'}^g = j_{k'}\right) \tag{42}$$

$$\overset{\text{D}}{=} \alpha_2 \sum_{i,j_1,\ldots,\not{j}_k,\ldots,j_{n_g}} A^g_{i,j_1,\ldots,j_{n_g}} \pi(V^g)_i \prod_{k'=1,\ldots,\not{k},\ldots,n_g} \lambda(U_{k'}^g)_{j_{k'}}. \tag{43}$$

Here $1,\ldots\not{k}\ldots,n_g$ stands for the integers from 1 to n_g with the exception of k. The index k' generally ranges over these integers except in the numerator of Eq. (40) where it covers the full range $1,\ldots,n_g$. The constants α_1 and α_2 are given by

$$\alpha_1 = \frac{1}{\Pr\left(e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g}\right)} \quad \text{and} \quad \alpha_2 = \frac{\Pr(e^+_{V^g})}{\Pr\left(e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g}\right)}, \tag{44}$$

but since these constants are independent of j_k and since we know that the vector $\pi(U_k^g)$ is normalized, they can simply be determined as normalizing constants.

Thus it is clear that even in the presence of statistical dependence among the daughter nodes, the λ and π vectors can be propagated in a fairly straightforward fashion.

4.4. Learning the conditional probabilities

We will now turn to the problem of learning the conditional probabilities stored in the matrices from training data. A training sample is an instantiation of the observed nodes of a network. The training problem is that of choosing the link matrices such that the overall probability of the training material (i.e. the product of probabilities of each sample given the link matrices) is maximized. We will distinguish two cases: the simple case in which all nodes are observed and the more complicated case involving matrices between unobserved nodes.

4.4.1. Case 1: all nodes are observed

Suppose we wish to find the link matrix $\Pr(Y \mid X)$ between two nodes X and Y. In this case we simply instantiate a matrix of counters $(C(X,Y)_{ij})$ which counts the number of times node X is in state i and node Y is in state j. After processing the entire training data and updating the respective counters, the maximum likelihood estimate of the matrix $\Pr(Y \mid X)$ is then given by

$$\Pr\left(Y=y_j \mid X=x_i\right)=\frac{C(X,Y)_{ij}}{\Sigma_{j'}C(X,Y)_{ij'}}. \tag{45}$$

Now consider the slightly more complicated case in which a node V has n daughters structured in G groups, $U_1^1,U_2^1,\ldots,U_{n_1}^1,U_1^2,\ldots,U_{n_2}^2,\ldots,U_{n_G}^G$, that was discussed in Section 4.1. Here the relationship between the parent V and the daughters $U_1^g,U_2^g,\ldots,U_{n_g}^g$ in group g is expressed in the tensor shown in Eq. (14). If the nodes $V,U_1^g,U_2^g,\ldots,U_{n_g}^g$ are all observed we can again simply instantiate a tensor of counters $C(V,U_1^g,\ldots,U_{n_g}^g)_{ij_1\ldots j_{n_g}}$ and count the number of occurrences of each event over the training data. The maximum likelihood estimate for A^g is then

$$A^g_{i,j_1,\ldots,j_{n_g}}=\frac{C(V,U_1^g,\ldots,U_n^g)_{ij_1\ldots j_{n_g}}}{\sum\limits_{j_1',\ldots,j_{n_g}'} C(V,U_1^g,\ldots,U_n^g)_{ij_1'\ldots j_{n_g}'}}. \tag{46}$$

4.4.2. Case 2: some nodes are unobserved

If the nodes involved are unobserved, it is no longer possible to count simultaneous occurrences in a straightforward way. However, we can ask for the expected number of simultaneous occurrences. For a single sample e this expected number equals the probability

$$\Pr\left(V=i,U_1^g=j_1,\ldots,U_n^g=j_{n_g} \mid e\right). \tag{47}$$

Here again, we split the daughter nodes into different groups according to their dependencies and only consider one group as it is independent of all the others. Fortunately, there is an easy way to calculate the quantity in Eq. (47) using intermediate results from the belief propagation. We have

$$\Pr\left(V=i,\, U_1^g=j_1,\ldots,U_n^g=j_{n_g} \mid e\right)$$

$$\overset{\mathrm{B}}{=}\Pr(V=i \mid e)\ \Pr\left(U_1^g=j_1,\ldots,U_n^g=j_{n_g} \mid V=i,\ e\right) \tag{48}$$

$$\overset{\mathrm{D}}{=}\mathrm{BEL}(V)_i\ \Pr\left(U_1^g=j_1,\ldots,U_n^g=j_{n_g} \mid V=i,\ e_{\bar{V}^g}\right) \tag{49}$$

$$\overset{(9),\mathrm{B}}{=}\alpha\lambda(V)_i\pi(V_i)\frac{\Pr\left(e_{\bar{V}^g} \mid V=i,\ U_1^g=j_1,\ldots,U_n^g=j_{n_g}\right)\ \Pr\left(U_1^g=j_1,\ldots,U_n^g=j_{n_g} \mid V=i\right)}{\Pr\left(e_{\bar{V}^g} \mid V=i\right)} \tag{50}$$

$$\overset{(27),\mathrm{D}}{=}\alpha\prod_{g'=1}^{G}\lambda(V^{g'})_i\pi(V)_i\frac{\Pr\left(e_{\bar{V}^g} \mid U_1^g=j_1,\ldots,U_n^g=j_{n_g}\right)A^g_{i,j_1,\ldots,j_{n_g}}}{\lambda(V^g)_i} \tag{51}$$

$$= \alpha \prod_{g'=1,\ldots,g,\ldots,G} \lambda(V^{g'})_i \pi(V)_i \ \Pr\left(e^-_{U_1^g},\ldots,e^-_{U^g_{n_g}} \mid U_1^g = j_1,\ldots,U_n^g = j_{n_g}\right) A^g_{i,j_1,\ldots,j_{n_g}} \tag{52}$$

$$\overset{\text{D.S}}{=} \alpha\pi(V^g)_i \prod_{k=1}^{n_g} \Pr\left(e^-_{U_k^g} \mid U_k^g = j_k\right) A^g_{i,j_1,\ldots,j_{n_g}} \tag{53}$$

$$\overset{\text{D}}{=} \alpha\pi(V^g)_i \prod_{k=1}^{n_g} \lambda(U_k^g)_{j_k} A^g_{i,j_1,\ldots,j_{n_g}}. \tag{54}$$

Instead of counting joint occurrences, we now add the expected number of joint occurrences in the tensor $C(V,U_1^g,\ldots,U^g_{n_g})$. Hence after processing all samples e in the data we obtain

$$C\left(V,U_1^g,\ldots,U^g_{n_g}\right) = \sum_e \Pr\left(V,U_1^g,\ldots,U^g_{n_g} \mid e\right), \tag{55}$$

where the sum denotes tensor addition. An estimate for the conditional probability tensor A^g can now be obtained using Eq. (46). It should be noted that since Eq. (47) represents the expected number of joint occurrences for one sample, Eq. (54) represents the expected number of joint occurances over the entire training data.

Note however that a previous estimate of A^g is required in Eq. (54). Hence, unlike the "all nodes observed" case the tensors can only be learned iteratively. We therefore have the following learning algorithm:

1. Choose initial (perhaps random) connection tensors.
2. Process the training data once, accumulating the tensor $(\Pr(V=i,U_1^g=j_1,\ldots,U_n^g=j_{n_g} \mid e))$ in a "counting" tensor $C(V,U_1^g,\ldots,U_n^g)$, for each parameter tensor in the network.
3. Normalize the C tensors and obtain the new estimates for the A tensors, viz.

$$A^g_{i,j_1,\ldots,j_{n_g}} = \frac{C(V,U_1^g,\ldots,U_n^g)_{ij_1\ldots j_{n_g}}}{\sum\limits_{j'_1,\ldots,j'_{n_g}} C(V,U_1^g,\ldots,U_n^g)_{ij'_1\ldots j'_{n_g}}}. \tag{56}$$

4. Go back to step 2 until convergence is achieved.

We will show in the next section that re-estimating the parameters in this way is guaranteed to increase the overall likelihood of the training data. Hence this iterative technique converges to a "local maximum likelihood" estimator. Like in the Baum–Welch algorithm, convergence to the true maximum likelihood estimator is not guaranteed.

The entropy calculated over the training data can be used as an indicator to decide when convergence is achieved. It is advisable to smooth this over several iterations and discontinue training when the smoothed entropy no longer decreases significantly.

5. A theorem about convergence of the iterative training method

In this section we will show that the previously described method for updating the probability estimates of the link tensors always improves the overall likelihood of the training data. In order to state the theorem we need to do some preliminary work.

Fig. 7 shows three causal trees. The nodes of the trees are represented as circles and squares. The squares represent *evidence nodes*, i.e. nodes for which the state can be observed. The circles represent unobserved nodes. The triangles represent connection tensors. As can be seen, the daughters of a given parent are sometimes assumed to be marginally independent (when each link has its own tensor) and sometimes exhibit dependencies (when certain links share tensors). The trees are said to model the

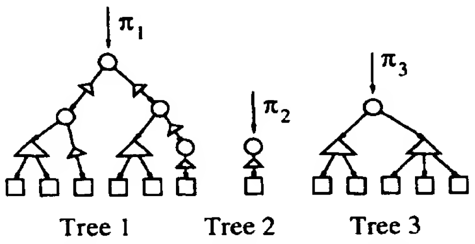


Fig. 7. Three isolated trees spanning part of the observation sequence.

observed nodes. The parameters of the model consist of the components of each of the connection tensors as well as the prior vectors of the root nodes (π_1, π_2 and π_3 in Fig. 7).

For simplicity we assume here that the evidence nodes coincide with the leaf nodes of the tree. The theorem that follows does not require this, but it makes the discussion easier. Evidence nodes are instantiated by providing their λ vector. In the simple (deterministic) case this vector has the form $(0,\dots,0,1,0,\dots,0)$ indicating by the position of the 1 the value of that node. It may however be any vector of non-negative real numbers. We could for example use the vector of word likelihood values calculated by a speech recognizer for each word of the recognizers vocabulary.

An instantiation of the evidence nodes is called a sample. For a given sample, we can calculate the probability of this sample given the model parameters. To do this recall that the λ and π vectors at a node X were defined as

$$\lambda(X) = \Pr(e_X^- \mid X), \tag{57}$$

$$\pi(X) = \Pr(X \mid e_X^+). \tag{58}$$

Thus

$$\lambda(X)\cdot\pi(X) = \sum_i \Pr(e_X^- \mid X=i)\Pr(X=i \mid e_X^+) \tag{59}$$

$$= \sum_i \Pr(e_X^- \mid X=i, e_X^+)\Pr(X=i \mid e_X^+) \tag{60}$$

$$= \Pr(e_X^- \mid e_X^+). \tag{61}$$

Eq. (60) is justified, for the node X separates e_X^- from e_X^+ and so the probability of e_X^- only depends on the state of X. At the root node of a tree, $e_X^+ = \emptyset$, so $\lambda\cdot\pi = \Pr(e)$, where e stands for the part of the evidence spanned by the tree. When there are multiple trees as in Fig. 7, the fact that these are not connected implies that they span independent parts of the sample. Thus the overall probability of the sample is obtained by multiplying the terms $\lambda(R)\cdot\pi(R)$ over all root nodes R.

It is usually more convenient, however, to use the negative logarithm of this quantity. This is known as the entropy of the sample. We define the entropy of a tree T with root node R as

$$E_R(T) = -\log(\lambda(R)\cdot\pi(R)), \tag{62}$$

and the entropy of the sample as

$$E(T) = -\sum_{\text{root nodes } R} \log(\lambda(R)\cdot\pi(R)). \tag{63}$$

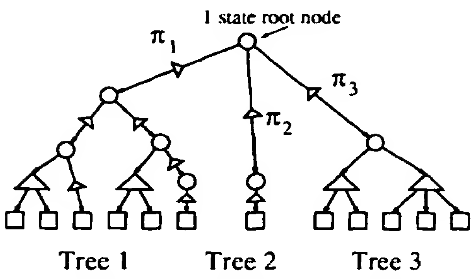


Fig. 8. Combination of the three isolated trees of Fig. 7 into a single one.

5.1. Parameterization of the model

Fig. 8 shows how the isolated trees of Fig. 7 can be converted into a single equivalent tree. This is done by introducing a new root node R and connecting the previous root nodes as daughters to this node. The new root node represents a random variable that can only take one value (thus it is not random at all). The components of the connection matrix connecting R to one of the old root nodes R_i are set identical to the components of the prior π_i of R_i in Fig. 7 (since R can only take one value the connection matrix is really just a vector). We do not need to consider the prior to R, for since R only takes one value this prior is always equal to the unit vector (1).

It should thus be clear that Fig. 8 is equivalent to Fig. 7. The advantage of this transformation is that we only need to consider one tree and that prior vectors are represented as connection tensors, so a re-estimation formula for the connection tensors will automatically apply to the priors as well.

The total set of parameters to be estimated thus consists of the components of the various connection tensors in Fig. 8.

Weight linkage. Since the latter part of this paper makes extensive use of a concept known as weight linkage we will introduce this concept here. We may impose additional constraints on a model by forcing certain connection tensors to have identical components. Thus if two tensors A^x and A^y of the model have the same rank (number of indices) and corresponding indices have the same dimension we can impose the additional constraint:

$$A^x_{ij_1\ldots j_n} = A^y_{ij_1\ldots j_n} \quad \text{for all} \quad i, j_1, j_2, \ldots, j_n. \tag{64}$$

This reduces the number of free parameters of the model.

In the following we will consider different models, which although having the same geometrical structure (or topology) differ in the actual parameter values. To differentiate between two such models we will use the letter Θ to indicate one parameter set and $\overline{\Theta}$ to indicate another. Thus the probability of a sample e under model Θ will be written as $\Pr(e \mid \Theta)$ and that of e under model $\overline{\Theta}$ will be written as $\Pr(e \mid \overline{\Theta})$. More specifically Θ is defined as the set of all connection tensors of the model and we will write $A \in \Theta$ to indicate that a given tensor A is part of model Θ. Naturally Θ can be partitioned into a set of subsets

$$\Theta = \bigcup_h \Theta^h, \tag{65}$$

where each subset Θ^h contains all tensors that are linked. (Linking of tensors really defines an equivalence relation on Θ and Θ^h are the equivalence classes.) In slight abuse of notation we will also write

$$(A : V, U_1, \ldots, U_n) \in \Theta^h, \tag{66}$$

to indicate that tensor A, which links the parent node V to the daughter nodes U_1 to U_n belongs to the tensors in Θ^h.

We can now state the main result of this section.

Theorem 1. *Let Θ be a parameter set for a model describing samples e of a training data set* Train. *Let E_Θ be the entropy of the training data given the parameter set Θ. Then if we choose a new parameter set $\overline{\Theta}$ for the same model which decomposes equally into subsets $\overline{\Theta}^h$ of linked tensors such that for any tensor $\overline{A} \in \overline{\Theta}^h$ we have*

$$\overline{A}_{ij_1\ldots j_n} = \frac{C^h_{ij_1\ldots j_n}}{\sum\limits_{j'_1,\ldots,j'_n} C^h_{ij'_1\ldots j'_n}}, \tag{67}$$

where

$$C^h_{ij1\ldots j_n} = \sum_{e \in \text{Train}} \sum_{(A:V,U_1,\ldots,U_n) \in \Theta^h} \Pr\left(V=i,\ \bigwedge_k U_k = j_k \middle| e, \Theta\right) \tag{68}$$

and denote $E_{\overline{\Theta}}$ the entropy of the training data given $\overline{\Theta}$ we have

$$E_{\overline{\Theta}} \leqslant E_\Theta. \tag{69}$$

The proof of this theorem is given in Appendix A.

The usefulness of Theorem 1 should be clear. The partial contributions

$$\Pr\left(V=i,\ \bigwedge_{k=1}^{n} U_k = j_k \middle| e, \Theta\right) \tag{70}$$

to the C^h tensor can readily be calculated using Eq. (54). After processing the training data once, one merely needs to re-normalize the C^h tensors to obtain a new estimate of the model parameters. The theorem guarantees that the new set of parameters models the training data better than the previous one.

6. An application: learning the hidden structure of language

The previous sections have been quite general in describing how probabilistic inferences can be made given a certain model (i.e. a graph) and also how the parameterization of the model can be learned even if some nodes are not observed. We will now turn to a specific application in which the developed ideas will be used.

The particular problem studied is that of language modeling, i.e. providing a probabilistic model that assigns probabilities to sequences of which correspond to their relative frequencies. Such models are useful in speech recognition where they assist the speech recognizer by favoring likely word sequences over unlikely ones.

The specific model adopted could be regarded as lying somewhere between the popular n-gram language models and a stochastic context-free grammar. The grammatical information is stored in the form of stochastic re-write (or produciton) rules, i.e. in rules of the form

$$i \rightarrow \alpha\ (p), \tag{71}$$

where i is a non-terminal symbol and α is a string of non-terminal and terminal symbols and p is the probability of this rule being used given that i is a symbol to be rewritten. However, unlike a genuine context-free grammar, the observation sequence is not divided into sentences prior to parsing. Instead,

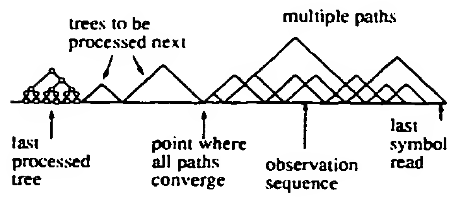


Fig. 9. This diagram illustrates the parsing algorithm utilizing the partial backtracking technique. When a new symbol is read all (newly) possible parse trees are constructed bottom up. This defines a lattice of trees. The possible paths through the lattice are traced and pruned such that only the best (i.e. minimal entropy) path for each end point is kept. At some earlier point (relative to the symbol just read) all these paths converge, identifying a unique sequence of trees. These trees are then chosen for the propagation and re-estimation algorithm.

the algorithm finds its own segmentation of the observation sequence and builds parse trees separately for each segment. When applied to natural language such segments are usually smaller than sentences, but they may also be as large or even larger than the actual sentences. In short, the model is ignorant of sentence boundaries, such as would be denoted by full stops in the observation sequence.

In the model rule productions such as (71) are viewed as causal relationships, the left-hand side causing the right-hand side. In this framework we will see that the ideas developed in the previous sections are directly applicable and allow us to not only parse the observation sequence but also learn the production rule probabilities from example text.

6.1. Overview of the algorithm

To accomplish our task the model has to perform two tasks:

1. The selection of the best segmentation points and the structure of the parse tree (the structure problem).
2. The calculation of the probabilities of the various terminal and non-terminal symbols in the parse trees (the assignment problem).

It is clear that if the structure problem was somehow solved for us, we could use the theory of Bayesian trees developed earlier to solve the Assignment problem by a suitable propagation of λ and π vectors. The re-estimation of grammar parameters could similarly be performed at the same time. Since, however, we are not given the answer to the structure problem we need to solve both problems simultaneously. At outline of this procedure will now be discussed.

Fig. 9 shows a partially parsed observation sequence. Symbols are processed one by one left to right as they come in. Over the observation sequence a number of parse trees are built (and later pruned). Some of these are shown in the right-hand side part of Fig. 9. These trees form a lattice over the observation sequence and it remains to pick the best sequence of trees, i.e. the one that maximizes the likelihood of the observation sequence. This could be solved by dynamic programming if we were giving definite start and end points of this lattice. In our problem there are no such endpoints, as we are processing an infinite string of words. If, however, we impose a maximum on the size of the trees [1], we can select trees among the best path after only processing a finite amount of the data using a technique known as partial backtracking (Brown et al., 1982).

[1] By the size of a parse tree we mean the number of terminal symbols it spans.

6.2. A specific grammar model

To solve the structure problem we need to essentially search through the space of all possible parse trees and select the most suitable. In order to keep the search space as small as possible we will require the grammar to be in Chomsky Normal Form. [2]

It is well known that any context-free grammar can be written in Chomsky Normal form (by increasing the number of non-terminal symbols if necessary), so the Chomsky normality constraint does not impose any structural constraints on the language, but only on the parse trees by requiring these be essentially binary, each node having two daughters except for the so-called pre-terminal nodes which have only one. It also allows us to represent the grammar in the following quantities: a tensor of rank 3, a matrix and a vector. We write

A_{ijk}: the probability of non-terminal symbol i re-writing to the two non-terminals j and k,
B_{im}: the probability of the non-terminal i re-writing to the terminal symbol m, and
P_i: the prior probability of the first symbol in a derivation (the root).

These three quantities must satisfy the stochastic constraints:

$$\sum_{jk} A_{ijk} + \sum_{m} B_{im} = 1 \text{ for all } i \tag{72}$$

and

$$\sum_{i} P_i = 1. \tag{73}$$

For a given parse tree, we can now apply the belief propagation algorithm. The tensors A and B serve as connection tensors and P provides the prior to the root node of the tree. In fact, any given tree will contain many instances of the A and B tensors, one for each terminal and non-terminal production, respectively, all of which are linked. The non-terminal nodes are all unobserved and are best thought of as random variables that may take the non-terminal symbols as values. Similarly each terminal node represents a random variable ranging over all terminal symbols. These terminal nodes are all observed. They are instantiated from the observation sequence by assigning them λ vectors of the form $(0,\dots,0,1,0,\dots,0)$ (called indicator vectors), where the position of the i indicates the identity of the terminal symbol.

6.3. Convergence of the training method

In Section 5 we showed that the iterative training method for the model parameter converges. However, this theorem was posed under the assumption that the tree structure connecting the nodes is given and fixed. So the question arises whether the same result is applicable to the situation described above, where this structure is chosen dynamically and may be different from iteration to iteration.

A moments thought will reveal that a sufficient condition for its applicability is the fact that trees are selected on the basis of maximizing the overall likelihood of the training data. To see this suppose that in a given iteration the tree topology T_1 is chosen as optimal. Then Theorem 1 guarantees that the likelihood of the data for this topology will increase. The likelihood for other topologies may increase or decrease as the case may be. If in the next iteration another tree topology T_2 has an even higher

[2] In Chomsky Normal Form (Chomsky, 1959) the only rules that are allowed are the ones for which α in equation production either equals a string of two non-terminal symbols or else is a single terminal symbol.

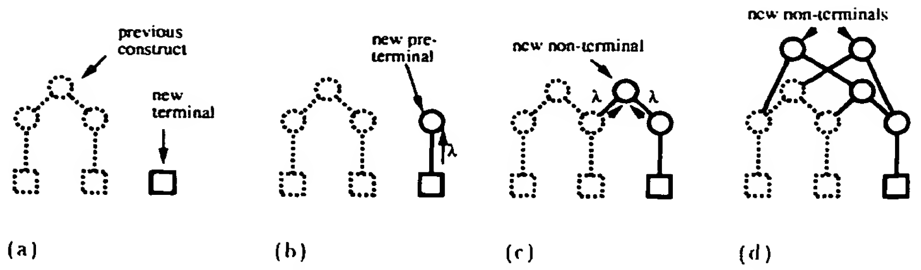


Fig. 10. The tree building (parsing) algorithm.

likelihood than that of T_1, it will become the preferred topology. So in any case the likelihood of the data in the new iteration will not have decreased.

6.4. Tree building methods

As was explained above, a lattice of trees is constructed during the parsing of the utterance, from which the best sequence is selected. We will now turn to the problem of constructing the tree lattice.

6.5. Tree construction: version 1

Since the grammar model is constrained to be in Chomsky Normal Form, the topologies of the trees are fairly constraint. In particular each terminal symbol has a unique "pre-terminal" node as its parent. These nodes may be created immediately, as soon as the terminal symbol has been observed (Fig. 10(a) and (b)). Next the λ vector, which has been instantiated for the terminal node can be propagated up to the pre-terminal node. The general propagation equations were given in section extensions (Eqs. (19) and (27)), but in this simple case these collapse to a simple matrix–vector multiplication:

$$\lambda(U) = B\lambda(X), \tag{74}$$

where U is the pre-terminal node and X is the terminal node. The new vector $\lambda(U)$ is independent of the shape of the eventual parse tree, so it can be calculated at this stage and need not be calculated again (Fig. 10(b)).

The newly created pre-terminal node can be regarded as a tree of size one, and we can calculate the entropy of this tree using Eq. (62). Next, for two neighboring pre-terminal nodes, a non-terminal node is proposed (Fig. 10(c)), having the two pre-terminals as daughters. Again, the λ vector may be passed up this new node. The propagation equation follows again from Eqs. (19) and (27) and in this case read

$$\lambda(V)_i = \sum_{jk} A_{ijk}\lambda(U_1)_j\lambda(U_2)_k, \tag{75}$$

where V is parent node and U_1 and U_2 are the two daughters. It is not yet clear whether this non-terminal will be part of the final tree, but if it is then again the its λ vector will not change as it depends only on its daughters. Again we can calculate the entropy of this tree (of size 2) using Eq. (62). If the entropy calculated is larger than the sum of the entropies calculated previously for the two daughter nodes, the new node is rejected and removed from memory. Otherwise it is kept as a potential node for the final tree. The process continues now in the same fashion by constructing new non-terminal nodes between any pair of existing non-terminal nodes which span adjacent sequences of the observation sequence (Fig. 10(d)).

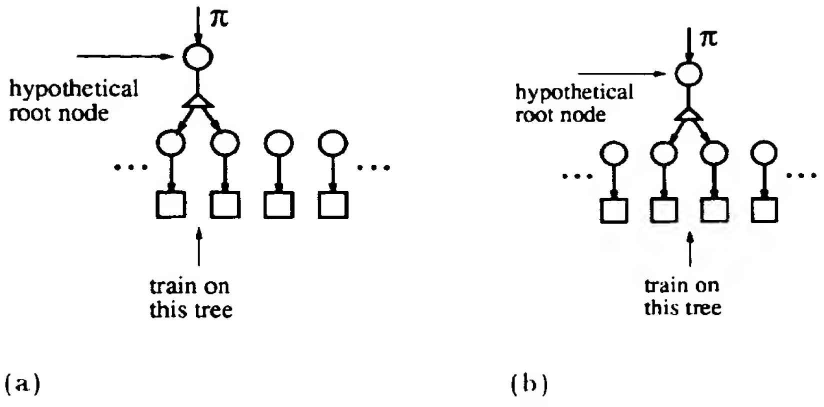


Fig. 11. Illustration of version 2 of the training algorithm.

Continuing this procedure leads to a lattice of nodes. While trees are being constructed they can also be pruned using the partial backtracking technique mentioned earlier.

6.6. Problems with version 1 of the tree construction algorithm

The algorithm above has a serious deficiency that must be rectified before the algorithm can be used in practice: it is not capable of learning the succession of two non-terminal symbols unless they occur in the same tree. In particular during the first iteration the grammar parameters are still random and each tree has size 1. Therefore, with the above training method as it stands the probabilities of the A tensor cannot be learned.

By modifying the tree building algorithm we can however avoid this problem.

6.7. Tree construction: version 2

In version 2 of the tree construction algorithm, trees are constructed and isolated in the same way as in version 1. However, prior to calculating the partial contributions to the weight re-estimation term (Eq. (55)) two neighboring trees are combined into a large tree by hypothesizing a common root node. Fig. 11(a) illustrates this, during the first iteration, when all constructed trees have size 1. After all weight contributions have been calculated, the hypothetical root node is removed and the next two trees are combined to a new tree (Fig. 11(b)). (The tree who formed the right branch of the previous construct now forms the left branch.) Training is repeated on this new construct. In this way, the information about neighboring trees is learned by virtue of the hypothetical nodes, and hence "tree-growth" is possible during training.

6.7.1. Allowing multiple priors

Version 2 of the tree building algorithm depends upon the initial symbol distribution vector P. An additional reduction in entropy can be achieved by allowing this vector to vary according to size of the tree to which it is applied.

In this scenario, rather than storing just one vector P, we use an array of vectors prior[s]. For a tree of size s (i.e. which spans s symbols of the observation sequence) the vector prior[s] is used as the prior for that tree.

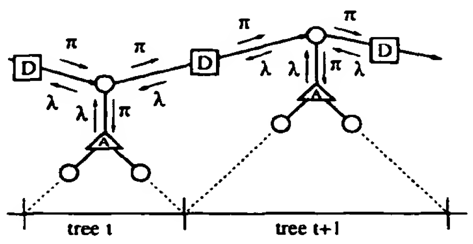


Fig. 12. Illustration of version 3 of the training algorithm.

6.8. Tree construction: version 3

Version 3 of the tree building algorithm is very similar to version 2. To understand it let us first consider an extension to the stochastic context free grammar formalism.

Up to now the grammar was described by three quantities: B_{im} describing the terminal productions, A_{ijk} describing the non-terminal productions and P_i describing the prior probability of the root symbol. We now replace the prior P_i by a matrix of conditional probabilities D_{ij}, describing the value of the current root node given the value of the previous root node. To describe this more precisely, if we have a sequence of trees $T_1, \ldots$ spanning the observation sequence, with root nodes $R_1, \ldots,$ then

$$D_{ij} = \Pr(R_t = j \mid R_{t-1} = i). \tag{76}$$

One way of looking at this is to say that we provide a bigram grammar for the root nodes of the trees. We now no longer have isolated trees, but rather one large structure. The λ and π vectors can be propagated through this structures just as the theory dictates (Section 4.1). Each root node has two groups of daughter nodes. One group consists of the two immediate daughters within the same tree. Its relation to the parent is described by the tensor A. The other group consists of a single node, the root node of the next tree, and the relation is described by the matrix D.

Fig. 12 illustrates how the λ and π vectors are propagated through the network.

If this architecture is used for training, the D matrix learns the "root node bigrams". However, this knowledge has to be transferred into the A tensor, if the trees are to grow in subsequent iterations. Otherwise we are in a situation similar to the one of version 1 of the tree construction algorithm.

In principle this problem can be solved by linking the parameters of the D matrix with those of the A tensor. But since they have different dimensions this cannot be achieved in quite such a straightforward way. However, the entries of the D matrix can be calculated from the entries of the A tensor by re-introducing the initial symbol vector P.

In version 2 of the tree building algorithm we hypothesized a new root node H for any pair of adjacent trees with root nodes R_1 and R_2. Thus, the tensor A learns the conditional probabilities $\Pr(R_1, R_2 \mid H)$. Since the prior of H is given by the initial symbol distribution vector P, we can get the joint distribution of $\Pr(R_1, R_2)$ by multiplying A by P:

$$\Pr(R_1 = j, R_2 = k) = \sum_i A_{ijk} P_i, \tag{77}$$

and from this we can calculate the conditional probabilities by simple normalization:

$$\Pr(R_1 = j \mid R_2 = k) = \frac{\Pr(R_1 = j \mid R_2 = k)}{\sum_{j'} \Pr(R_1 = j' \mid R_2 = k)}. \tag{78}$$

Hence we can calculate D from A by setting

$$D_{jk} = \frac{\sum_{i} A_{ijk} P_i}{\sum_{i,j'} A_{ij'k} P_i}. \tag{79}$$

The tree-building and learning algorithm can now be described as follows. At the beginning of the iteration the D matrix is calculated from the A tensor and the P vector according to Eq. (79) above. Trees are then constructed using version 1 of the tree building algorithm. The root nodes are then linked using the D matrices as shown in Fig. 12 and the λ and π vectors are propagated through the entire network. When it comes to calculating the contributions to the re-estimation terms (Eq. 47) the D matrices are again replaced by the A tensors and the λ vectors from the two root nodes plus the P vector are used to calculate the contribution to the A tensor at this point of the structure.

6.9. *Discussion on tree building algorithms*

In the previous subsections we have presented three closely related algorithms for parsing the observation sequence. Of these, version 1 does not work during the training period, because the tree are not encouraged to grow. (It is however a useful algorithm once the grammar parameters have been learned.) The main reason for including it here was its simplicity and the fact that it forms the basis for the other two algorithms.

Version 2 is a simple extension of version 1, that does allow trees to grow from iteration to iteration. The drawback of this algorithm is that training is no longer performed on the structure exhibiting minimal entropy, but rather on a modification of this structure. The fact that the minimal entropy structure has been modified implies that the convergence argument given in Section 6.3 does no longer strictly apply. So the overall entropy calculated over the observation data may no longer be strictly decreasing from iteration to iteration. However, experiments have confirmed that this is not a problem in practice.

Version 3 is a neat extension of version 2 which extends the grammar model by a bigram grammar between root nodes, so that the trees are no longer independent of each other and in the process provides a justification for the hypothetical root nodes.

For brevity, we refer to the three algorithms collectively as BLI (Bayesian Language Inference) algorithms.

One may ask why sentence-end information is ignored in the algorithms. We believe that Bayesian networks are very flexible tools and many information sources can be incorporated if and when they become available. For example, if partial bracketing information is available, the tree structures could be forced to be consistent with these. If the identity of certain non-terminal symbols are known, the corresponding nodes could be clamped to these values. Long distance semantic or pragmatic information could be made available in the form of priors.

Here, we present the algorithm in its purest form. The fact that it does not require sentence boundaries distinguishes it from other approaches and is regarded an asset. Many text databases are not provided with explicit sentence-end markers. (Periods occur at the end of a sentence but also elsewhere.) If the algorithm is to be used in an application, all available knowledge sources could, and indeed should, be realized.

6.10. *Interface with a speech recognizer*

In this paper we only apply the model to symbolic sequences. However, we will briefly describe how the algorithm can be naturally integrated with an HMM type speech recognition front end processor.

The Viterbi decoding algorithm (or the forward calculation) produces likelihoods of the form $P(O \mid M)$ of the probability of the acoustic observation O given the (Markov) model M. This probability

Table 1
Comparison to the Inside–Outside algorithm

	Inside–Outside	BLI model
Inside probabilities	$e(s, t, i) = \sum_{jk} \sum_{r=s}^{t-1} A_{ijk} e(s, r, j) e(r+1, t, k)$	$\lambda(u)_i = \sum_{jk} A_{ijk} \lambda(v)_j \lambda(x)_k$
Outside probabilities	$f(s, t, i) = \sum_{jk} \left(\sum_{r=1}^{s-1} A_{jki} f(r, t, j) e(r, s-1, k) + \sum_{r=t+1}^{T} A_{jik} f(s, r, j) e(t+1, r, k) \right)$	$\pi(v)_j = \alpha \sum_{ik} \pi(u)_i \lambda(x)_k$
Sentence boundaries	required	not required
Tree size	is sentence length	not fixed (grows during training)
Complexity	cubic in the sentence length cubic in number of non-terminals	linear cubic in number of non-terminals [3]
Initial symbol	always "S" prohibits tree growth	any non-terminal

corresponds directly to the diagnostic probabilities $\lambda(s_t) = P(e_{s_t}^- \mid s_t)$. Here the evidence $e_{s_t}^-$ is identical to the acoustic observation O and the terminal symbol s_t corresponds to the model M. Thus, when probabilistic information like the calulation of an HMM is available, we can substitute the indicator vector by a vector containing the HMM likelihoods as the λ vector of the leaf nodes.

6.11. Relation to the Inside–Outside Algorithm

The Inside–Outside Algorithm (Baker, 1979; Lari and Young, 1991) is a similar algorithm that is capable of inferring stochastic context-free grammars in Chomsky Normal Form from example text. Table 1 shows the main differences between this algorithm and the method reported here.

We consider the Bayesian framework described in the first part of the paper more flexible than the Inside–Outside algorithm. Not only does it lead to simpler propagation equations by just considering one parse tree rather than all, it is also considerable more general with language modeling as just one application. In particular, it allows a number of variations of the basic language model such as the inclusion of a bigram grammar between the root nodes of the trees as was described in version 3 of the tree building algorithm. Of course the BLI algorithm could also easily be changed to honor sentence boundaries in the same manner as the Inside–Outside algorithm does.

However, we are here not concerned with the linguistic analysis of a sentence but with the provision of a stochastic language model for speech recognition. For this purpose the limited tree size may be entirely adequate, while training larger trees will require considerably more training material, consume more processing time per terminal symbol and runs the risk of producing sub-optimal solutions due to the added degree of complexity.

6.12. The complexity of the algorithm

The complexity of the algorithm is cubic in the number of non-terminal symbols [3] and linear in the length of the observation sequence. In comparison, the complexity of the Inside–Outside Algorithm is

[3] An improved version which is only quadratic in complexity in the number of non-terminals has recently been developed (Lucke, 1994).

cubic in both the number of non-terminals and the sentence length. We found experimentally the following average times on a DEC-station 5000/240 for processing one symbol of the observation sequence:

Number of non-terminals	Training	Testing
30	0.63 sec	0.57 sec
50	3.09 sec	2.74 sec

These training times were measured during the end of the training. At the beginning, processing is considerably faster as the trees are much smaller.

Since the training procedure is iterative, the required number of parameter updates until convergence is also a crucial factor in determining the algorithms complexity. We know of no theoretic result that asymptotically describes the rate of convergence. Experimentally we found that on noisy data [4] about 100 parameter updates were required.

7. Experimental work

7.1. Database

The experiments were based on the the ATR Dialogue database. This is a text database consisting of some 8500 Japanese sentences of telephone conversations together with their part of speech labels (Ehara et al., 1990). The database contains a total of about 6500 different words with 51 different parts of speech. The model is capable of learning stochastic re-write rules from an unlabeled sequence of symbols. However, the number of terminal and non-terminal symbols needs to be chosen in advance. To ensure accurate estimation of the production probabilities it is required that each non-terminal symbol occurs reasonably frequent. For this reason we used the part of speech symbol sequence rather then the word sequence as the input to the model. In this way enough training material for each input symbol was ensured.

For the purpose of the experiment the data base was divided into two equal portions, one for training and one for testing.

7.2. *When to perform parameter updates*

In HMM training, it is customary to update the HMM probabilities each time the entire training data has been processed. In the experiments reported here we update more frequently. Thus, successive training epochs are carried out on different part of the training data. This is motivated by the following observation: the grammar inference mechanism described here only requires an unlabeled source of symbols to operate. Even though training is performed on a finite amount of training data for the experiments described here, it is feasible that the algorithm could be used on an infinite symbol source. At present when the end of the training data is reached the algorithm starts again at the beginning of the training data. With an infinite symbol source one could indefinitely perform incremental improvements to the weights by continuously processing the output of the source.

[4] The term "noisy data" here means that different training epochs are performed on different parts of the training data.

Initially when the model parameters are still random, only trees of size 1 are constructed (see experimental section and Fig. 14). During this phase essentially the uni-gram statistics of the data base are learned. It follows that it is not necessary to have a very long update period during this time of learning. As the sizes of the trees increases, the model parameters can only be accurately estimated by taking more and more data into account during one training epoch. Thus it seems advantageous to start with a relatively small update period and increase this gradually. This was done in the experiments, starting with an update period of 100 symbols, which was increased by 30 after each update.

7.3. Comparison of tree building methods

In the first experiment the three parsing algorithms were compared. For this purpose we trained models with 30 non-terminal symbols. The results are as follows:

Method	Tree size	Train	Test
Version 1	1.0	4.48	4.46
Version 2	2.81	2.81	2.80
Version 3	2.94	3.88	3.87

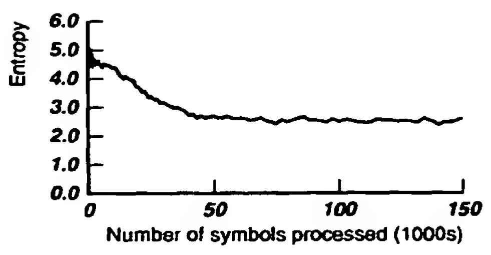
The table shows the average size of the trees constructed by the algorithm and the entropy over the test and training data. As was pointed out earlier, version 1 of the parsing algorithm does not lead to tree growth and was included as a control.

Somewhat surprisingly version 2 performed significantly better than version 3, despite the fact that version 3 provides a bigram type grammar for the root nodes of the trees.

7.4. Evaluation of the BLI model

In order to evaluate the performance of the BLI model, a version with 50 non-terminal symbols were trained. The number 50 was chosen for two reasons: firstly it roughly agrees with the number of terminal symbols (51) and secondly it was the largest number that could be trained in reasonable time on a workstation. Following the results of the previous section we used version 2 of the tree building algorithm. Other model parameters were chosen as follows:

Number of terminals	51
Number of non-terminals	50
Initial update period	100
Update period increase	30
Maximum tree size	6



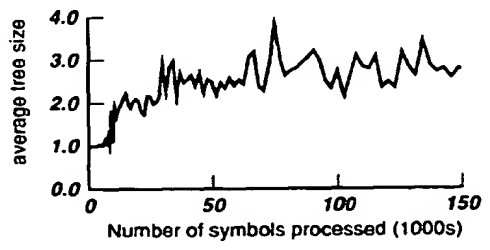


Fig. 13. Development of the entropy duFigg14aiThegaverage tree size during training.

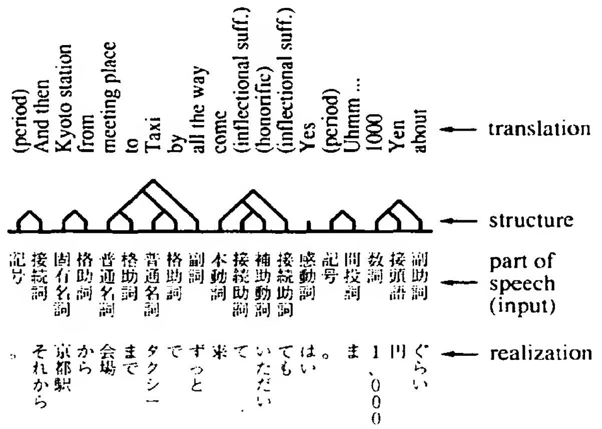


Fig. 15. Example of parsed text using the algorithm. The text roughly translates to "And then you take a taxi from Kyoto station all the way to the meeting place. It will be about 1000 yen".

Table 2
Experimental results

Model entropy	Bigram	Trigram	BLI
Training data	2.83	2.39	2.60
Test data	2.84	2.76	2.70

Fig. 13 shows the development of the entropy when estimated during training and Fig. 14 shows the average tree size during each training epoch. Initially the entropy was close to 6, but dropped sharply to the "unigram-level" of about 4.5 after the first few iterations. During this period the tree size was 1. When trees started to be constructed the entropy dropped further. Both curves are quite noisy. This is mainly attributed to the fact that different training epochs were carried out on different parts of the training material.

Training was discontinued after 300,000 symbols had been processed. The trained model was subsequently evaluated on the test data and an entropy of about 2.7 bits was observed. This was compared to the entropies obtained from a bigram or trigram grammar [5]. The overall results are summarized in Table 2.

As can be seen, the BLI model outperforms the bigram model, but its entropy is similar to the trigram entropy. On the training data, the BLI entropy is considerable higher than the trigram entropy. The difference in entropy between the training and test data is also smaller for the BLI model. This means that the generalization is higher for the BLI model and should scale better to larger tasks once the problems concerning training time have been overcome.

Fig. 15 shows how the BLI algorithm processed part of the training material. It shows the part of speech sequence that was used as input to the model together with the parse trees constructed. The underlying word sequence and a translation was included as a reference.

[5] The n-gram grammars were smoothed using a constant floor level that was obtained from "held-out" data.

A visual inspection revealed that the parse trees tended to correspond to Japanese grammatical units such as the *bunsetsu*. [6] However, in many cases a non-alignment was also observed. In particular since the sentence end marker (period) is processed just like any other symbols, it is also incorporated in the tree structures. Occasionally such a tree combines the end of one sentence with the beginning of the next, contrary to common linguistic analysis.

8. Discussion

In this paper we have shown that parse trees may be regarded as Bayesian networks. This framework enables one to integrate rule probabilities of stochastic grammars and observational uncertainties (such as acoustic match likelihoods) in a mathematically sound and computationally efficient way. Moreover, we have shown that the causal and diagnostic support vectors that arise as intermediate quantities in the Bayesian formalism may be used, in a natural way, to calculate new grammar parameter estimates. The parameter re-estimation is guaranteed to increase the likelihood of the observed data, as long as the parse trees are selected on a maximum likelihood principle.

The main advantage of this procedure over similar algorithm such as the Inside–Outside algorithm lies in its flexibility. In this paper we have exploited this flexibility by building a language model that learns grammatical constructs "bottom-up", i.e. is not forced (or limited) to recognize whole sentences. We have also shown how a bigram grammar across the root symbols can be integrated easily and naturally and how probabilistic scores of an HMM could naturally be utilized.

In the future it seems feasible that additional knowledge sources such as semantic or pragmatic information could be integrated to further assist the speech recognition and parsing process. Already Bayesian networks are used to represent semantic knowledge in probabilistic expert systems. By formulating the parsing mechanism in the same language a powerful integration of various knowledge sources may result.

The building and selection method is currently not truly satisfactory. Firstly it only considers the best sequence of trees (an invitation to local (i.e. non-global) maxima). Also it was shown that the maximum likelihood selection criterion (required in the convergence proof) had to be sacrificed in order to allow tree growth. This difficulty is reflected in the AI domain where at current Bayesian Networks are handcrafted – no truly automatic process exists.

Despite the simplicity and weaknesses of the parsing algorithm we were able to demonstrate experimental results which were superior to the established bi- and trigram grammars. The better generalization ability that was observed is also notable.

It is however important to consider the limitations of the technique. These are mainly due to the computational load incurred. The algorithm's complexity is cubic in the number of non-terminals [7], limiting the number of non-terminal symbols that can be used. In this paper we have considered discrete sequences of words. This is applicable for isolated word recognition systems, i.e. systems in which the word boundaries are known. Processing continuous speech using the BLI model is in principle possible: one would need to construct the tree lattice on a frame-by-frame (rather than word-by-word) basis. Even in relatively efficient schemes such as the one proposed by Ney (1991), this leads to a significant increase in the computational load. Another possibility would be to employ the method as a post-filter in order to re-order a list of candidate word strings found by a speech recognizer. This could be done relatively efficient using a stack decoder (Jelinek, 1969; Paul, 1991), although the A^* search procedure would have

[6] Bunsetsu are phrase-like constructs which are the dominant syntactic category in Japanese.
[7] But see Footnote 3.

to be replaced by a non-admissable search to allow the processing of larger contexts (Paul, 1992). It is our opinion that this difficulty for processing continuous speech mainly arises from the fact that the Viterbi algorithm can essentially only incorporate regular grammars (such as n-gram). An algorithm for decoding HMM that is more tolerant towards more general type grammars would be desirable.

Acknowledgements

I would like to thank S. Sagayama for giving me the freedom to pursue this topic, Y. Sagisaka for his continued support and Paul Taylor for proof reading an earlier version of this paper. Also the two reviewers have provided many constructive comments, for which I am grateful.

Appendix A. Proof of Theorem 1

This proof is a generalization of the corresponding one for Hidden Markov Models (Huang et al., 1990). Both learning procedures are applications of the more general EM algorithm (Dempster et al., 1977). We will first consider the case in which the training data Train consists of just one sample e. Let $\mathcal{N}$ denote the total set of nodes in the model and let

$$S : \mathcal{N} \mapsto \mathbb{N} \tag{80}$$

be a function that assigns a particular value to each node in the model and let S be the set of all such functions. Further for the sample e and a value allocation S we denote

$$\Pr(e,\ S \mid \Theta) = \Pr\left(e,\ \bigwedge_{U \in \mathcal{N}} U = S(U)\ \middle|\ \Theta\right). \tag{81}$$

A moments thought will reveal that

$$\Pr(e,S \mid \Theta) = \prod_{(A:V,U_1,\dots,U_n) \in \Theta} A_{S(V),S(U_1),\dots,S(U_n)} \prod_{\text{leaf nodes } X} \lambda(X)_{S(X)}. \tag{82}$$

Now for two different sets of parameters Θ and $\bar{\Theta}$ define the function

$$Q(\Theta,\bar{\Theta}) = \frac{1}{\Pr(e \mid \Theta)} \sum_S \Pr(e,S \mid \Theta) \log \Pr(e,S \mid \bar{\Theta}), \tag{83}$$

and consider the term $E_\Theta - E_{\bar{\Theta}}$. By the concavity of the log function we have

$$E_\Theta - E_{\bar{\Theta}} = \log \frac{\Pr(e \mid \bar{\Theta})}{\Pr(e \mid \Theta)} \tag{84}$$

$$= \log\left(\sum_S \frac{\Pr(e,S \mid \Theta)}{\Pr(e \mid \Theta)} \frac{\Pr(e,S \mid \bar{\Theta})}{\Pr(e,S \mid \Theta)} \right) \tag{85}$$

$$\geqslant \sum_S \frac{\Pr(e,S \mid \Theta)}{\Pr(e \mid \Theta)} \log \frac{\Pr(e,S \mid \bar{\Theta})}{\Pr(e,S \mid \Theta)} \tag{86}$$

$$= Q(\Theta,\bar{\Theta}) - Q(\Theta,\Theta). \tag{87}$$

It follows that a sufficient condition for $E_{\bar{\Theta}} \leqslant E_{\Theta}$ is that $Q(\Theta,\bar{\Theta}) \geqslant Q(\Theta,\Theta)$. This latter condition is clearly satisfied for a $\bar{\Theta}$ that maximizes $Q(\Theta,\cdot)$. In fact we have strict inequality:

$$E_{\bar{\Theta}} < E_{\Theta}, \tag{88}$$

unless Θ itself maximizes $Q(\Theta,\cdot)$. We therefore ask for the set of parameters $\bar{\Theta}$ that maximizes $Q(\Theta,\cdot)$ for a given Θ. From Eq. (82) we have

$$\log \Pr(e,S \mid \bar{\Theta}) = \sum_{(\bar{A}:V,U_1,\dots,U_n)\in\bar{\Theta}} \log \bar{A}_{S(V),S(U_1),\dots,S(U_n)} + \sum_{\text{leaf nodes } X} \log \lambda(X)_{S(X)}. \tag{89}$$

Substituting this into Eq. (83) and changing the order of summation gives

$$Q(\Theta,\bar{\Theta}) = \sum_{\bar{\Theta}^h \subset \bar{\Theta}} \sum_{i,j_1,\dots,j_n} C^h_{ij_1\dots j_n} \log \bar{A}_{ij_1\dots j_n} + \sum_{\text{leaf nodes } X} \sum_i D(X)_i \log \lambda(X)_i, \tag{90}$$

where the tensor $\bar{A}_{ij_1\dots j_n}$ is the representative tensor of the class $\bar{\Theta}^h$,

$$C^h_{ij_1\dots j_n} = \sum_S \sum_{(A:V,U_1,\dots,U_n)\in\Theta^h} I(S(V)=i) \prod_{k=1}^{n} I(S(U_k)=j_k) \frac{\Pr(e,S \mid \Theta)}{\Pr(e \mid \Theta)} \tag{91}$$

$$= \sum_{(A:V,U_1,\dots,U_n)\in\Theta^h} \sum_{S \mid S(V)=i, \wedge_k S(U_k)=j_k} \Pr(S \mid e,\Theta) \tag{92}$$

$$= \sum_{(A:V,U_1,\dots,U_n)\in\Theta^h} \Pr\left(V=i, \bigwedge_{k=1}^{n} U_k=j_k \,\middle|\, e,\Theta\right) \tag{93}$$

and

$$D(X)_i = \sum_S I(S(X)=i) \Pr(S \mid e,\Theta) \tag{94}$$

$$= \Pr(X=i \mid e,\Theta) \tag{95}$$

$$= \mathrm{BEL}_{\Theta}(X)_i. \tag{96}$$

The second term in Eq. (90) is independent of $\bar{\Theta}$, so it remains to maximize

$$\sum_{\bar{\Theta}^h \subset \bar{\Theta}} \sum_{i,j_1,\dots,j_n} C^h_{ij_1\dots j_n} \log \bar{A}_{ij_1\dots j_n}, \tag{97}$$

with respect to the components of the representative tensor $\bar{A}_{ij_1\dots j_n}$ of $\bar{\Theta}^h$ subject to the constraints

$$\sum_{j_1,\dots,j_n} \bar{A}_{ij_1\dots j_n} = 1. \tag{98}$$

It is easy to show (for example by using Lagrangian multipliers) that this maximization is in fact solved by

$$\bar{A}_{ij_1\dots j_n} = \frac{C^h_{ij_1\dots j_n}}{\sum_{j'_1,\dots,j'_n} C^h_{ij'_1\dots j'_n}}. \tag{99}$$

This completes the proof of the Theorem for the special case of there being just one sample e in the training set. In the case of n samples, we simply create one large sample by writing all n samples one after each other. We use a model which consists simply of n copies of the model for one sample, with the corresponding tensors linked across all copies. This large model applied to the large sample is equivalent to the original model applied repeatedly to all samples in the training data. Carrying out the same analysis as above for the large model leads to Eqs. (67) and (68) as claimed.

References

J.R. Anderson (1981), "A theory of language acquisition based on general learning principles", *Proc. 7th Internat. Conf. on Artificial Intelligence, Vancouver, BC*, pp. 97–103.

L.R. Bahl, P.F. Braown, P.V. de Souza and R.L. Mercer (1989), "A tree-based statistical language model for natural language speech recognition", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 37, No. 7.

J.K. Baker (1979), "Trainable grammars for speech recognition", *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, ed. by D.H. Klatt and J. Wolf, pp. 547–550.

R. Berwick (1980), "Computational analogues of constraints on grammars: A model of syntax acquisition", *Proc. 16th Annual Meeting of the Association for Computational Linguistics and Parasession on Topics in Interactive Discourse, Pennsylvania.*

P.F. Brown, J.C. Spohrer, P.H. Hochschild and J.K. Baker (1982), "Partial traceback and dynamic programming", *Proc. Internat. Conf. Acoust. Speech Signal Process.*, pp. 1629–1632.

N. Chomsky (1959), "On certain formal properties of grammars", *Information and Control*, Vol. 2, pp. 137–167.

R.G. Cowell, A.P. Dawid and D.J. Spiegelhalter (1993), "Sequential model criticism in probabilistic expert systems", *IEEE Pattern Anal. Machine Intell.*, Vol. 15, No. 3, pp. 209–219.

A.P. Dempster, N.M. Laird and D.B. Rubin (1977), "Maximum likelihood from incomplete data via the EM algorithm", *Royal Statistical Society Ser. B*, Vol. 39, pp. 1–38.

T. Ehara, N. Inoue, H. Kohyama, T. Hasegawa, F. Shohyama and T. Morimoto (1990), Contents of the ATR Dialogue Database, Technical Report TR-I-0186, ATR Interpreting Telephony Research Laboratories.

K.-S. Fu and T.L. Booth (1986), "Grammatical inference: Introduction and survey – Part I and II", *IEEE Pattern Anal. Machine Intell.*, Vol. PAMI-8, No. 3.

M.E. Gold (1967), "Language identification in the limit", *Inf. and Control*, Vol. 10, pp. 447–474.

X.D. Huang, Y. Ariki and M.A. Jack (1990), *Hidden Markov Models for Speech Recognition*, Edinburgh Univ. Press, Edinburgh.

F. Jelinek (1969), "A fast sequential decoding algorithm using a stack", *IBM J. Res. Develop.*, Vol. 13.

F. Jelinek (1991), "Up from trigrams!", *Proc. Eurospeech, Genova, Italy*, pp. 1037–1040.

K. Lari and S.J. Young (1991), "Applications of stochastic context-free grammars using the Inside–Outside algorithm", *Computer Speech and Language*, Vol. 5, pp. 237–257.

S.L. Lauritzen and D.J. Spiegelhalter (1988), "Local computations with probabilities on graphical structures and their application to expert systems", *Statistical Society*, pp. 157–224.

H. Lucke (1994), "Reducing the computational complexity for inferring stochastic context-free grammar rules from example text", *Proc. Internat. Conf. Acoust. Speech Signal Process., Adelaide, Australia*, pp. I-353–356.

J. McClelland and D. Rumelhart (1987), "A parallel distributed processing model of aspects of language acquisition", in *Mechanisms of Language Acquisition*, Lawrence Erlbaum, Hillsdale, NJ.

H. Ney (1991), "Dynamic programming parsing for context-free grammars in continuous speech recognition", *IEEE Trans. Signal Process.*, Vol. 39, No. 2, pp. 336–340.

D.B. Paul (1991), "Algorithms for an optimal A* search and linearizing the search in the stack decoder", *Proc. Internat. Conf. Acoust. Speech Signal Process.*, pp. 693–696.

D.B. Paul (1992), "An efficient A* stack decoder for continuous speech recognition with a stochastic language model", *Proc. Internat. Conf. Acoust. Speech Signal Process.*, pp. I-25–28.

J. Pearl (1987), *Probabilistic Reasoning in Intelligent Systems – Networks of plausible inference*, Morgan Kaufmann, San Mateo, CA.

J.G. Wolff (1980), "Language acquisition and the discovery of phrase structure", *Lang. Speech*, Vol. 23, pp. 255–269.

J.G. Wolff (1982), "Language acquisition, data compression and generalization", *Lang. Commun.*, Vol. 2, No. 1, pp. 57–89.